ACTIVE WINDOW MANAGEMENT FOR REORDER BUFFER

Prior Foreign Application

[0001]   This application claims priority from European patent application number 00108698.2, filed April 20, 2000, which is hereby incorporated herein by reference in its entirety.

Technical Field

[0002]   The present invention relates to storage devices in computer systems and in particular, it relates to an improved method and system for operating system storage devices, and in particular to buffer devices which are used in a circulating manner.

Background Art

[0003]   Although the present invention has a broad field of application, as improving or optimizing buffer storage strategies is a very general purpose in computer technology, it will be described and discussed with prior art technology in a special field of application, namely in context of utilization of an instruction window buffer, further referred and abbreviated to IWB which is present in most modern computer systems in order to enable a parallel program processing of instructions by a plurality of processing units. Such processors are referred to herein as out-of-order processors.

[0004]   In many modern out-of-order processors such a buffer is used to contain all the instructions and/or register contents before the calculated results can be committed and removed from the buffer. When results were calculated speculatively beyond the outcome of a branch instruction, they can be rejected once the branch prediction becomes wrong by cleaning these entries from the buffer and overwriting them with new correct instructions. This is one prerequisite for the out-of-order processing. One main parameter influencing the performance of the processors is the buffer size: A big buffer can contain many more instructions and results and therefore allows more out-of-order processing. One design objective therefore is to have a big buffer. This, however, stays in conflict with other design requirements such as cycle time, buffer area, etc.

[0005]   When, for example, the buffer size is dimensioned too large then the efforts required to manage such a large plurality of storage locations decreases the performance of the buffer. Furthermore, increased buffer size implies an increased signal propagation delay. Thus, generally, any performance-improved buffer storage method has to find a good compromise between the parameters buffer size, storage management and therewith storage access speed.

[0006]   In US patent No. 5,584,037 titled 'Entry Allocation In A Circular Buffer', which is hereby incorporated herein by reference in its entirety, the instructions stored in a reservation station used like the

before mentioned IWB are addressed via a bitstring where the 1 to 0 and 0 to 1 transitions of the active window bit stream indicate the beginning and the end of the active window. The active window bit is ON when an entry contains valid data. Otherwise it is switched OFF. The IWB is a circular buffer hence all entries containing valid data are consecutive and therefore the transition of the active window bit from 0 to 1 and 1 to 0 identifies the in- and out-pointer as long as at least one entry is kept free. When the buffer is empty (no active bit at all) an arbitrary entry is written.

[0007]   The disadvantage lies in the fact that for performance purposes, this prior art way of operating such a buffer based on a serialization of reading or writing the IWB with respective determination of the respective state of each instruction is too slow, in particular, when each entry must be accessible to a plurality of read/write requesters which define or read the state of the buffered entries, e.g., instructions. Furthermore one entry must be kept free in the prior art approach to assure that there is still a transition in the active window bits of the IWB. This reduces the utilization of the IWB.

Summary of the Invention

[0008]   It is thus an objective of the present invention to increase the efficiency of buffer utilization, i.e., to increase its performance.

[0009]   This objective of the invention is achieved by
the features stated in enclosed independent claims. Further
advantageous arrangements and embodiments of the invention
are set forth in the respective subclaims.

[0010]   According to basic features of the present
invention an "active bit" is associated to each IWB entry
and the state of this active bit is generated by
combinatorial logic associated with the entry. Thus, a bit
vector is generated. Each active bit represents a
concentrated form of entry-related validation information
which is evaluable for the status of each entry relative to
the further processing of the entry by the one or more
processes accessing the buffer entries. The presence of an
active window bit vector prevents the necessity to check
sequentially for validity of the instruction.

[0011]   The state of these active bits is generated based
on the flow of instructions in the buffer that update the
in-pointer and out-pointer value.

[0012]   A second, preferred inventional aspect is based
on a new approach to decentralize the computing work
required for evaluating the validation information (AWB) of
the entries, i.e., to provide for autonomous determination
of the relevant status information by the respective entry
itself. The approach stands in a sharp contrast of any prior
art buffer management for managing the desired access to
read requesters or from write requesters which traditionally
reads the required control information from multiple

locations of the buffer, makes a synthesis and an evaluation
of the control information at a central location by a
dedicated processing unit and uses the evaluation results
for 'remote controlling' the respective plurality of buffer
entry accesses.

[0013]   The inventional approach, however, saves data
transfers and complexity of the overall processing because a
simple additional circuit is added to the buffer itself
which automatically generates the active window status
information required for the plurality of processes like
renaming registers, issuing and committing instructions, as
an output associated with a respective entry and
automatically generated when an IN- and OUT pointer pair
specific for each of the plurality of processes is input in
the circuit. In particular, the automatic status generation
is very advantageous against prior art where the new status
information had to be derived from the status information of
the preceding status stored in latches because of cycle time
requirements.

[0014]   Each entry stores its actual buffer index. By a
logical circuit comprising a comparison between the index
and the respective relation to the respective current IN and
OUT pointer values an entry is made 'intelligent' in the
sense that it knows if it belongs to the valid entries for
which the active window bit needs to be ON, i.e. entries
between the OUT and IN pointer with possible wrap-around.

[0015]   According to a preferred aspect of the present invention this is basically achieved by providing and managing validation information specific for each of the k processes and indicating if a respective entry can be subjected to a respective process, or not. This is done preferably by providing for each entry a circuit comprising combinatorial logic which automatically calculates the status for the respective process to do or already done.

[0016]   Thus, a novel method is disclosed in which the active bits are generated cellular for each IWB entry. Each cell contains a greater-equal compare that is used to calculate, based on an in- and an out-pointer, if the entry is part of the active window. Thereby different in- and out-pointer values are applied for the different IWB macros to match the active window to the macro protocol requirements. As a further advantage there are no undetermined cycles because the validation information can be obtained before the end of the cycle in which a respective value pair of the IN- and OUT pointer is input to the combinatorial logic.

[0017]   Generally, the inventional concepts are applicable for any buffer management implementing piece of hardware, and in particular for wrap-around buffers, as well. Further, any buffer used for queue management can be improved by applying the inventional concepts.

Brief Description of the Drawings

[0018]   The present invention is illustrated by way of examples and is not limited by the shape of the figures of the accompanying drawings in which:

[0019]   Fig. 1 is a schematic representation showing basic aspects and elements used during the inventional method according to a specific embodiment thereof applied to operation of an instruction window buffer,

[0020]   Fig. 2 is a schematic representation showing further aspects and illustrating elements used during the inventional method according to a preferred embodiment thereof applied to operation of an instruction window buffer, and

[0021]   Fig. 3 is a block diagram illustrating a combinatorial logic circuit used for generating entry-related validation information.

Best Mode for Carrying Out the Invention

[0022]   IWBs typically hold 16 to 64 instructions in today's implementations. The number will grow in future implementations since it is preferable to hold as much instructions as possible. However in the majority of the time only a fraction of the complete buffer will be filled

up with "active" entries, i.e., valid instructions to be executed and not yet ready to be overwritten by a new entry.

[0023]   The determination if an entry belongs to the "active window" is dependent on many processes as for example dispatching new instructions into the IWB, purging or partially purging the buffer after a mispredicted branch and retiring instructions. The active window is not uniform over the different fields and processes within the IWB. There are three states (Fig.2).

[0024]   1. Active window for renaming
This window spans all instructions from the youngest one dispatched, to the oldest instruction for which the result data has not yet been written back in the architected register (ARA).

[0025]   2. Active window for issue spans
from youngest instruction with renaming complete, to the oldest instruction not committed.

[0026]   3. Commit window spans
from youngest instruction dispatched, to the oldest instruction not committed.

[0027]   The active window increases within one cycle on the "IN" point by zero to the maximum of instructions dispatched to the buffer and decreases on the "OUT" point by zero up to the maximum of instructions committed. For efficiency the buffer is used in wrap around fashion.

[0028]   With general reference to the figures and with
special reference now to Fig. 1 a buffer 10 memory has a
plurality of 64 entries 0..63. The entries are indexed
subsequently according to their position in the buffer
array, i.e. entry 0 has the index 0, entry 1 has the index 1
etc. The indexes are bit strings comprised of the number of
bits necessary to indicate the binary value of the index.

[0029]   Each entry holds operation code data, source data
in respective fields 5, 6 and can be filled if required with
target data resulting from an execution of the associated
instruction. These are stored in a field 7 each. In
addition, further control and status information, not shown
in Fig.1 can be allocated to each entry.

[0030]   For each entry the buffer 10 comprises a
valid/invalid bit in a field 17. The total of them forms an
'active bit string' illustrated by a vertical extension in
the drawing. During operation active entries are
characterized by having the active window bit (AWB) switched
to ON, in the example '1' which can be seen in the very
first column on the left margin of the active window buffer
in Fig. 1.

[0031]   During program operation program instructions are
dispatched from a dispatching unit into the buffer 10. In
order to maintain a sequence of active entries without any
gaps between them a new instruction is entered into the
buffer 10 at the entry identified by the In-pointer index.
The entry location is marked by an IN-pointer 14.

[0032]   Correspondingly, an OUT-pointer 16 marks the
oldest instruction, i.e. the instruction which is to be
retired, i.e. removable from the buffer. This is the
location identified by the OUT-Pointer value. When the
In-Pointer and Out-Pointer have the same value the wrap bit
of the pointers will decide if the buffer is full or empty.
In case the wrap bit is set the buffer is full, in case the
wrap bit is not set the buffer is empty.

[0033]   When the result data of an instruction is read out
from the IWB and stored in the ARA the respective entry is
decided to be removable from the buffer 10, i.e., the entry
is left for being overwritten by the next one. Thus, the
active window bit is switched from 1 to 0. The state of
these active bits is thus updated based on the flow of
instructions in the buffer.

[0034]   According to the chosen case of IWB operation the
processes changing the active window bits are then dispatch,
commit, purge and partial purge.

[0035]   Under the assumption that an entry is active if its
active window bit = 1 the following information can be
advantageously be managed:

[0036]   1. Dispatch of new instructions to the IWB
The new instruction is dispatched to the location the
In-Pointer points to. If more than one instruction is
dispatched, the instructions are written in consecutive
order starting at the In-Pointer. The In-Pointer is

incremented by the number of new instructions dispatched to the IWB. In consequence the window bits of the new entries turn ON since their position is between the In-Pointer and the Out-Pointer.

[0037]   2. Retiring instructions from the IWB
The Out-Pointer points to the oldest non-committed entry in the buffer. The Out-Pointer is incremented by the number of instructions committed in the cycle. The window bits not belonging to the active instruction stream anymore are reset to zero since their position is smaller than the out-pointer.

[0038]   3. Complete Purge
In-Pointer = Out-Pointer = 0, Wrap=0. The result is that all window bits of the IWB turn to 0.

[0039]   4. Partial purge
The entry position (index) is sent together with the instruction to the execution units. In case the IWB has to be purged partially because of a mispredicted branch, the In-Pointer is set to the index following the entry of the mispredicted branch instruction. All window bits from the instruction following the mispredicted branch to the end of the window are reset to zero.

[0040]   5. IWB full
The IWB is full if In-Pointer=Out-Pointer and wrap=1. No action is required on window bits.

[0041]   With reference to Fig. 2 and according to a
preferred embodiment of the present invention and disclosing
a second preferred aspect thereof an instruction window
buffer 10 can be fed by instructions from the dispatching
unit 12 and feed them to a commit unit 13 after out-of-order
execution as it was the case in the example given before.

[0042]   In contrast thereto, however, three different bit
strings 20, 22 and 24 are maintained which serve to
determine the status of each instruction in view of a
respective one of three relevant processes which work on a
respective instruction each during out-of-order processing.
Those three different bit strings are referred to renaming
window 20, issue window 22, and commit window 24.

[0043]   The active window bit 17 which - in the example
given in Fig. 1 - has a quite general nature because it
refers generally to all three relevant processes, is now
split up into three different status bits each specifically
reflecting the progress of an instruction relative to the
respective specific process - renaming, issuing, and
committing. Thus, the general bit status 17 can be omitted
if not required by any other processing unit cooperating
with the buffer 10.

[0044]   Consequently, three independent pairs of In- and
Out-Pointers are the inputs for generating the specific
process windows 20, 22, and 24:
In/Out-Pointer_for_Commit[0..5],
In/Out-Pointer_for_Rename[0..5], and

In/Out-Pointer_for_Issue[0..5].

Some contents of the pointers may be identical, e.g., the In_Pointer_for_Issue and In_Pointer_for_Commit.

[0045] Thus, the basic aspect of the active window bit vector as described before is basically maintained but concurrently refined, i.e., sophisticated by additionally managing the same number of pointer pairs as there are processes working on the buffer entries contents.

[0046] Further, and with additional reference to Fig. 2 the vectors 20, 22, and 24 are not implemented as a latch chain which has to be written and to be read to maintain the information, but instead, the respective active/not-active bits associated with the three different bit vectors are generated cellular for each IWB entry by a respective combinatorial logic 30, 32, 34 which is illustrated exemplarily in Fig. 3 for three different entries 0, 1 and 2 - and for one respective window, as for example the issue window. The bit generation for the other windows is performed basically in the same way but is not explicitly shown in the drawing in order to improve its clarity.

[0047] The determination of the state of an entry is done by a greater-equal compare 36 and a less compare 38 of the physical entry position 35 with the In- and Out-pointer.

[0048] In the non-wrap-around case a given entry belongs to the active window if the entry position is greater or equal to the Out-pointer and smaller than the In-pointer.

The greater-equal compare 36 output is set to 1 for all
entries that are greater-equal than the Out-Pointer since it
compares the IWB entry number with the Out-Pointer value.
The less-compare 38 output is set to 1 for all IWB entries
that are smaller than the In-Pointer value since it compares
the entry number and the In-Pointer value. The output of the
AND gate 42 that has the compare outputs of 36 and 38 as
inputs will therefore be a '1' for all entries that are
between the In- and the Out-Pointer and therefore the
desired window bit string is generated on the outputs of the
OR gates 46.

[0049]   In the wrap-around case, the active entries are
all entries that are smaller than the In-Pointer and all
entries that are larger-equal to the Out-Pointer. Hence
these bits need to be set. The greater-equal compare again
generates a '1' at the output for all entries that are
larger-equal to the Out-Pointer. Furthermore, the
less-compare 38 generates a '1' on its output for all
entries that are smaller than the In-Pointer. So when the
output of 36 and 38 are ORed by 40 in each IWB entry and the
IWB-WRAP signal is ON for the AND gate 44, then the correct
window bit string is generated on the window bit outputs by
46.

[0050]   When the In-Pointer and the Out-Pointer are equal
the IWB-Wrap input defines if the IWB is full (case
IWB-Wrap=1) or empty (case IWB-Wrap=0). Hence all or none of
the active window bits have to be set. Since the compare
gates 36 and 38 now have the same input signals for each

entry and compare gate 36 is a greater-equal compare and the 38 compare is a less compare the output of 36 or 38 will be set - but never both are set at the same time. Hence the output of 42 will be '0' for each entry and the output of the OR 40 will be '1' for each entry. Hence if IWB-Wrap is '1' all window bits are set according to the IWB-Full case and when IWB-Wrap=0 all window bits are '0' according to the IWB-empty case.

[0051]   These comparisons are done in parallel for every entry.

[0052]   In more detail and with reference to the processes the entries are subjected to the manipulation of the state of the respective status bits is done in the following manner given for the IWB in a non-wrap case.

[0053]   1. Dispatch
The new instruction is dispatched to the location the In-pointer 14 points to. If more than one instruction is dispatched, the instructions are written in consecutive order starting at the In-pointer. Then, the In-pointer is incremented by the number of new instructions dispatched to the IWB, i.e., it is moved from top to down. In the In-wrap case, i.e., when the In-pointer window wraps around the In-pointer moves from the last entry back to the first entry and from there down again. In consequence the window bits of the renaming window of the new entries turn ON since their position is smaller than the In-pointer.

[0054]   The commit window in-pointer as the issue window in-pointer is incremented in the cycle when the entry is to be taken into account by the commit, respectively issue, process.

[0055]   2. Commit
The Out-pointer 16 points to the oldest non-comitted entry in the buffer. The Out-pointer is incremented by the number of instructions committed in the cycle, e.g., it is moved from top to down. In the Out-wrap-case the Out-pointer moves from the last entry back to the first entry and from there down again. The commit window bits not belonging to the active instruction stream anymore reset to zero since their position is smaller than the out pointer. The Out-Pointer for the rename window will include the committed instructions until they have been written by the commit process into the architectural register file ARA. For example, if this takes one cycle, then the Out-Pointer of the renaming window will be overwritten with the In-Pointer of the commit window with one cycle delay. The issue window Out-Pointer may be set to the Commit Out-Pointer value as well as the Rename window Out-pointer.

[0056]   3. Purge
For completely purging the buffer 10 the In-pointer value is set equal to the Out-pointer value, IWB-Wrap = 0. See the input signal IWB-Wrap in Figure 3, too. Then all window bits of the IWB turn to '0'.

[0057]   4. Partial purge
The entry position given by the above mentioned index is
sent together with the instruction to the execution units.
In case the IWB has to be purged partially because of a
mispredicted branch, the In-pointer is set to the index
following the entry of the mispredicted branch instruction.
All window bits from the instruction following the
mispredicted branch to the end of the window are reset to
zero.

[0058]   5. IWB full
The IWB is full if the value of the In-pointer equals that
one of the Out-pointer and IWB-Wrap = 1,
see Figure 3 for the IWB-Wrap signal. All window bits are
set.

[0059]   As should reveal from the foregoing description
the different windows 20, 22, and 24 are managed by applying
the associated pair of pointers to the specific compare
circuits. Thus, every entry "decides" by itself if it
belongs to the respective active window. This validation
update is done preferably in every cycle. Therefore the
desired entry validation information is available in the
same cycle the In- and Out-pointers are applied and is
available at each physical entry location immediately. Thus,
there are no undetermined cycles.

[0060]   In the foregoing specification the invention has
been described with reference to a specific exemplary
embodiment thereof. It will, however, be evident that

various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention as set forth in the appended claims. The specification and drawings are accordingly to be regarded as illustrative rather than in a restrictive sense.

[0061]   In particular, the splitting up to provide for a plurality of status windows can be combined or might not be combined with the advantageous feature of applying combinatorial logic as described above because they are independent from each other. Nevertheless, when combined, there is a synergy effect resulting because the additional computing work which would be per se required for handling the plurality of status bits is done in a very quick and simple way and in a decentralized manner.

[0062]   Further, when applied to buffers other than IWBs it should be understood that the number of active windows maintained is adapted to the number of performance relevant processes working on the buffer entries.

[0063]   The present invention can be included in an article of manufacture (e.g., one or more computer program products) having, for instance, computer usable media. The media has embodied therein, for instance, computer readable program code means for providing and facilitating the capabilities of the present invention. The article of manufacture can be included as a part of a computer system or sold separately.

[0064]   Additionally, at least one program storage device readable by a machine, tangibly embodying at least one program of instructions executable by the machine to perform the capabilities of the present invention can be provided.